MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines

Montek Singh and Steven M. Nowick

***Abstract*—An asynchronous pipeline style is introduced for high-speed applications, called MOUSETRAP. The pipeline uses standard transparent latches and static logic in its datapath, and small latch controllers consisting of only a single gate per pipeline stage. This simple structure is combined with an efficient and highly-concurrent event-driven protocol between adjacent stages. Post-layout SPICE simulations of a ten-stage pipeline with a 4-bit wide datapath indicate throughputs of 2.1–2.4 GHz in a 0.18-μm TSMC CMOS process. Similar results were obtained when the datapath width was extended to 16 bits. This performance is competitive even with that of *wave pipelines* [40], [19], without the accompanying problems of complex timing and much design effort. Additionally, the new pipeline gracefully and robustly adapts to variable speed environments. The pipeline stages are extended to fork and join structures, to handle more complex system architectures.**

***Index Terms*—Asynchronous, clocked CMOS, gate-level pipelines, latch controllers, micropipelines, pipeline processing, transition signaling, wave pipelining.**

I. Introduction

A NEW asynchronous pipeline style, called MOUSETRAP, is introduced for high-speed applications. The pipeline uses standard blocks of static logic for processing data and simple level-sensitive D-latches to separate data items.

An asynchronous, or clockless, circuit style [38] was chosen for several reasons. First, while synchronous designers are currently capable of achieving multi-gigahertz clock distributions, the task involves the ever-increasing challenges of design time, verification effort, clock skew, and power management, and interfacing with different timing domains. Second, since an asynchronous pipeline has no global clock, it has a natural *elasticity* [35]: the number of data items in the pipeline, and the speeds of the external interfaces, can vary dynamically. As a result, the pipeline can gracefully interface with environments operating at different rates, including those subject to dynamic voltage scaling, thus facilitating modular and reusable design. Finally, the localized control of asynchronous pipelines is an excellent match for very high throughput fine-grain datapaths.

The new pipeline is characterized by the simplicity of its structure and operation, as well as by ease of design. The datapath uses standard transparent latches which are small and fast, and, for a basic linear pipeline, the asynchronous control consists of only a single gate per pipeline stage. Pipeline stages communicate only with immediate neighbor stages, and the timing constraints are all local, simple, and one-sided.

While the proposed pipeline style has general applicability, a special focus of this paper is to target extremely high throughput. In particular, fine-grain, or "gate-level," pipelines are proposed, where the function logic in each stage is only one gate deep. At this granularity, very short cycle times are obtained: e.g., the critical cycle consists of a path through a single level of function logic plus two latch delays and a small amount of control logic. As an additional optimization, this critical cycle can be further shortened by merging together logic and storage elements, using a circuit style called *clocked-logic*, or *clocked-CMOS* (C^2MOS) [2]. In each case, a new, highly-concurrent protocol is used; as a result, a basic MOUSETRAP pipeline without logic has a cycle time of only 5–6 CMOS gate delays (three components).

The pipeline builds on, and extends, the more conservative approaches proposed in [7], [22], and [35]. In comparison, the MOUSETRAP pipeline generates an earlier completion signal, and new templates are proposed to handle complex pipelining (forks/joins). In addition, several novel optimizations are proposed: a "waveform shaping" strategy to speed up the critical path; an inverter elimination strategy using dual-rail control logic; and the use of a clocked-CMOS logic style.

The name MOUSETRAP stands for minimal-overhead ultra-high-speed transition-signaling asynchronous pipeline. There is another reason why our pipelines are so called: the latching action is somewhat analgous to that of a mousetrap. When a pipeline stage is waiting for data, its latch remains transparent; as soon as data enters the stage, it is captured by closing the latch behind it. While there have been other asynchronous pipelines that have used this kind of latching action [7], [35], each has its own limitations. In effect, our goal in this paper has been to build a "better mousetrap." Post-layout simulations using SPICE are quite encouraging: a 2.10–2.38 GHz[1] throughput in a TSMC 0.18-μm process.

The paper is organized as follows. Section II introduces the new pipeline, including its structure and operation, some performance-oriented optimizations, and extensions to handle forks and joins. Section III presents previous work on synchronous

Manuscript received June 18, 2004; revised June 22, 2006. This work was supported by the National Science Foundation (NSF) under Award CCR-97-34803, by the NSF ITR under Award CCR-00-86036, by the NSF ITR under Award CCR-00-86007, by a grant from the New York State Microelectronics Design Center (MDC), by a gift from Sun Microsystems, Inc., by a UNC Junior Faculty Development Award, by a grant from UNC University Research Council, and by an IBM Faculty Development Award.

M. Singh is with the Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599 USA (e-mail: montek@cs.unc.edu).

S. Nowick is with the Department of Computer Science, Columbia University, New York, NY 10027 USA (e-mail: nowick@cs.columbia.edu).

Digital Object Identifier 10.1109/TVLSI.2007.898732

[1]Strictly speaking, when referring to the throughput of asynchronous designs, the unit "gigahertz" should actually be interpreted as "giga items per second."

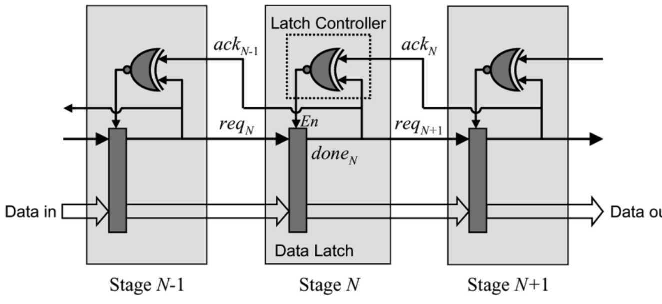

Fig. 1. Basic MOUSETRAP pipeline without logic processing.

and asynchronous pipelines, and then Section IV provides an in-depth comparison of MOUSETRAP with relevant previous approaches. Simulation results are presented in Section V and Section VI gives conclusions.

II. The MOUSETRAP Pipeline

This section first introduces the basic structure and operation of the MOUSETRAP pipeline (Sections II-A and II-B). Then, several implementation issues are discussed in detail and performance and timing constraints are derived (Sections II-C–II-E). In addition, an optimization is introduced that improves pipeline performance under steady-state operation by carefully "shaping" the controller output so as to reduce critical pipeline delays (Section II-F). Finally, the basic linear pipelines are extended to handle forks and joins (Section II-G).

Initially, to simplify discussion, Sections II-A and II-B focus on a basic pipeline without logic processing, i.e., a simple first-input, first-output (FIFO) queue. Later, Section II-C shows how logic processing is easily added.

A. Basic Pipeline Structure: A Simple FIFO

Fig. 1 shows the structure of the basic pipeline without logic processing. Three pipeline stages are shown. Each stage consists of a *data latch* and a *latch controller*. Adjacent stages communicate with each other using "requests" (req's) and "acknowledgments" (ack's).

The *data latch* is a standard level-sensitive D-type transparent latch. The latch is normally transparent (i.e., enabled), allowing new data to pass through quickly.

A commonly-used asynchronous scheme, called *bundled data* [26], is used to encode the datapath: a control signal, req_N, indicates arrival of new data at stage N's inputs. This approach, which has been successfully used in commercial chips by Philips [12], allows existing synchronous-style blocks to be reused in an asynchronous system without concerns for hazards, as long as the associated request signal is generated with appropriate timing. In particular, a simple one-sided timing requirement must be satisfied for correct operation: req_N must arrive *after* the data inputs to stage N have stabilized. (When logic processing is added to the pipeline, the request signal in each stage is typically delayed by an amount that matches the latency of the associated function block, i.e., by a *matched delay*. This is discussed in more detail in Section II-C.) Once new data has passed through stage N's latch, done_N is produced, which is sent to its latch controller, as well as to stages $N-1$ and $N+1$.

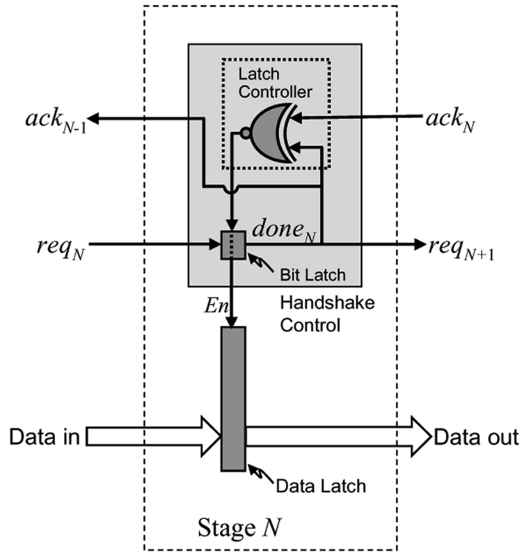

Fig. 2. Aternate view of a basic MOUSETRAP pipeline stage.

The *latch controller* enables and disables the data latch. It consists of only a single XNOR gate with two inputs: the done from the current stage, stage N, and the ack from stage $N+1$.

An alternate view of the basic pipeline is shown in Fig. 2. The latch inside a stage is shown separated into two parts: 1) a single bit latch that receives the incoming request req_N and produces done_N and the outgoing request req_{N+1} and 2) the remainder of the latch which captures the data bits. In this representation, the bit latch and the XNOR together form the entire control circuit that generates and receives the handshake signals from the neighboring pipeline stages on the left and the right, and also produces the latch enable signal En, which is internal to the stage, for controlling the latching action on the datapath.

B. Pipeline Operation

1) Overview: The operation of the pipeline of Fig. 1 is quite simple. Initially, when the pipeline is empty, all its latches are transparent and all the done, req and ack signals are low. When the first data item flows through successive stages of the pipeline, it flips the values of all these signals *exactly once* (high). Subsequently, the second data item flips all these signals once again (low). This method of signaling is called *transition signaling* [35]. Each transition, whether up or down, represents a distinct event: the arrival of a new data item.

Once a data item passes through stage N's latch, three actions take place *in parallel:* 1) the data is passed forward to the next stage for further processing, along with the corresponding request, req_{N+1}; 2) an acknowledgment, ack_{N-1}, is sent to the previous stage, freeing it up to process the next data item; and, finally, 3) stage N's latch itself is quickly made opaque to protect the current data from being overwritten by new data produced by stage $N-1$. Subsequently, when an acknowledgment, ack_N, is received from stage $N+1$, the latch in stage N is reenabled (i.e., made transparent).

Note that while transition signaling is used to signal the flow of data (one transition on each req/done/ack per data item) the latches themselves require *two transitions* per data item: one to capture (make opaque), and one to release (make transparent). The first transition takes place when data passes through stage N's latch (done_N changes value); the second when the same data passes through stage $N+1$'s latch (ack_N changes value). Thus, the XNOR gate acts like a *phase converter:* it converts the transition signaling done's and ack's into level control for the transparent latches.

There is another interpretation of the behavior of the latch controller, which is useful for understanding the pipeline operation: the XNOR gate is simply an "equality tester." When stages N and $N+1$ have the same data item, stage N is effectively "empty," and its latch is enabled (made transparent). When the stages have distinct data items, stage N is effectively "full," and its latch is disabled (made opaque).

The latching action by a pipeline stage is analogous to the operation of a household mousetrap: latches remain transparent *before* data arrives; they are closed (i.e., made opaque) as soon as data passes through. It is important to note that this behavior is very different from that of most synchronous, and many asynchronous, pipelines in which latches are opened only *after* new data arrives.

2) Detailed Operation: A key local timing constraint must be satisfied for correct operation. Since a transition on done_N is also a transition on ack_{N-1}, there is a race condition between the disabling of stage N's latch and the reception of new data from $N-1$. To ensure that the contents of stage N are not corrupted, stage N's latch must be disabled fast enough, i.e., before the stage $N-1$ can provide new data. This is a simple one-sided timing constraint that can easily be satisfied in practice. (For a more detailed analysis, see Section II-D2.)

Note that the choice of a hybrid protocol—*transition signaling* for the handshake signals, and *level signaling* for the latch enable signal—combines the advantages of both signaling schemes: 1) much less handshaking overhead, using transition signaling, since there is no wasteful "return-to-zero" phase and 2) use of small and fast transparent latches, since they are level controlled. The benefit of a single round-trip handshake per communication (two signaling events), compared to the more common two round-trip handshakes with four-phase communication (four signaling events), is especially important when function blocks communicate over long-latency interconnects, such as long-haul on-chip wires or off-chip buses. While several transition signaling schemes have been proposed—some with phase conversion [7], [35] and others without [42]—the pipeline presented here has much less overhead. (Refer to Section IV for a detailed comparison.)

The operation of a pipeline stage can be formally specified in the form of a signal transition graph (STG) [6], as shown in Fig. 3. Transitions $En+$ and $En-$ represent the enabling and disabling of the latch. The remaining STG transitions—req_N, req_{N+1}, ack_{N-1}, and ack_N—represent *transitions* (i.e., toggling) of the respective handshake signals. The directed arcs represent the dependencies between pairs of transitions. Thus, for example, En must be asserted high and the incoming request must transition before the outgoing request transitions. One arc,

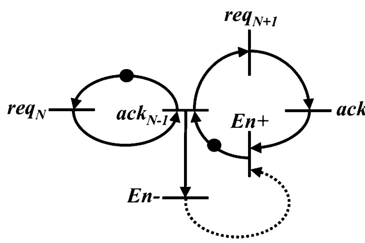

Fig. 3. Formal specification of stage controller.

from $En-$ to $En+$, is shown dotted; it represents a dependence that is not directly implemented by the controller, but instead must be satisfied by timing.

In summary, the new pipeline protocol is very simple and the operation quite fast. The forward latency of an empty pipeline is low because all the latches are initially transparent. The cycle time of the pipeline is short because the pipeline is highly concurrent: as soon as data enters stage N, stage $N-1$ is freed up for its entire next cycle.

C. Pipeline Implementation: Adding Logic Processing

So far, only pipelines without logic processing, i.e., simple FIFOs, have been considered. It is now shown how logic processing can easily be added to the pipeline.

1) General Pipeline Implementation: Fig. 4 shows how basic logic processing can be added to the pipeline. Blocks of combinational logic and matching delay elements ("matched delays") are simply inserted between pipeline stages. The standard asynchronous *bundled data* scheme is again used: req_N must arrive at stage N *after* the data inputs to that stage have stabilized. Therefore, the latency of the delay element must match the worst-case delay through the combinational block. A benefit of this approach is that the datapath itself can use standard single-rail (synchronous style) blocks, which are allowed to be hazardous, as long as the req arrives after data has stabilized. Moreover, even when worst-case matched delays are used, this approach has two advantages over synchronous design: 1) different pipeline stages are allowed to have different delays and 2) variations in a stage's latency are required to be tracked only by its local delay element, not by a global clock.

There are several common ways to implement a matched delay. One technique is to simply use an inverter chain, or a chain of transmission gates; the number of gates and their transistor sizing determines the total delay. A more accurate technique duplicates the worst-case critical path of the logic block, and uses that as a delay line [10]. If the duplicated critical path is placed in close proximity to the logic block, it can provide good delay tracking even for a wide variation in environmental and process variations. However, this technique is more area expensive than using a chain of inverters or other standard gates. Bundled data has been widely used, including in a commercial Philips 80C51 asynchronous microcontroller, several tens of millions of which have been sold commercially by Philips for use in pagers and smartcards [12]. Moreover, bundled data is also currently used in the commercial tool flow of Handshake

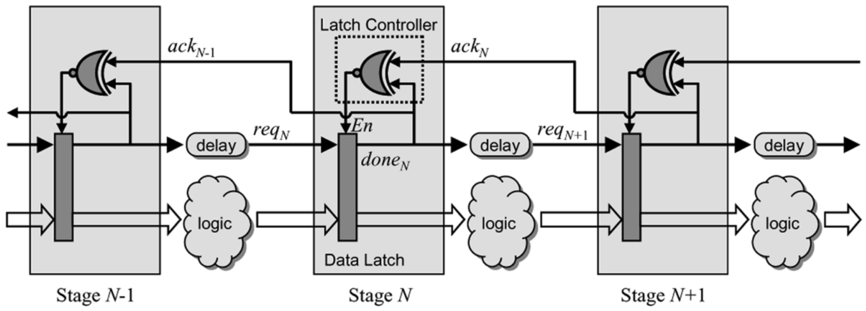

Fig. 4. MOUSETRAP pipeline with logic processing.

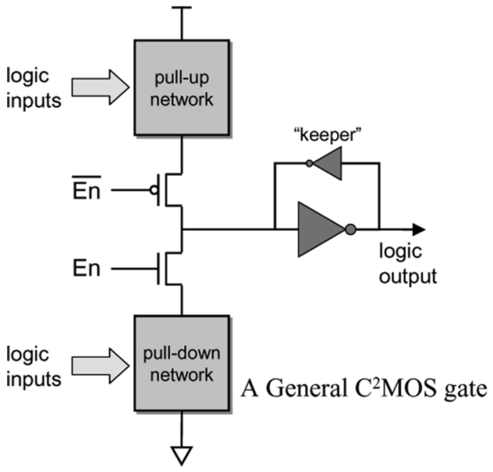

Fig. 5. C^{2}MOS logic.

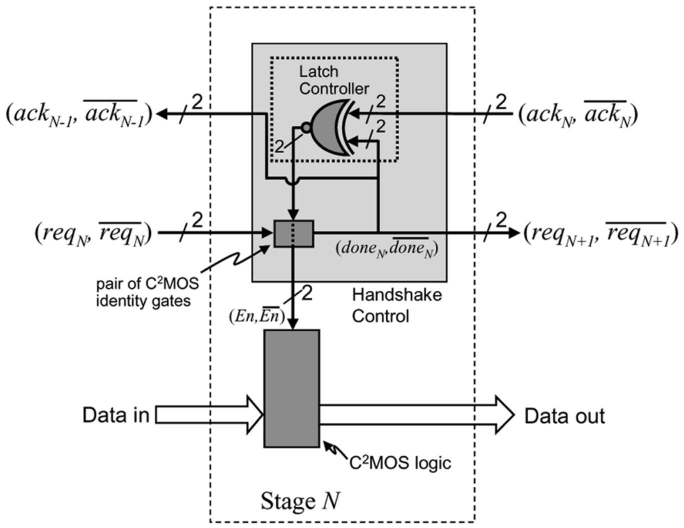
Latch Controller
$(ack_{N-1}, \overline{ack_{N-1}})$
$(ack_N, \overline{ack_N})$
$(req_N, \overline{req_N})$
$(req_{N+1}, \overline{req_{N+1}})$
$(done_N, \overline{done_N})$
pair of C²MOS identity gates
$(En, \overline{En})$
Handshake Control
Data in
Data out
C²MOS logic
Stage N
2

Fig. 6. C^{2}MOS implementation of gate-level MOUSETRAP pipeline stage.

Solutions [13], a Philips-incubated asynchronous startup company that, in partnership with ARM, has very recently developed a fully asynchronous bundled-data ARM family processor [1].

2) Special Case: Gate-Level Pipelines Using C^{2}MOS: To target extremely high throughput, as a special case, *gate-level pipelines* can be used: the datapath is sectioned into the finest-grained stages, each containing function logic that is only a single logic level deep, with no explicit latches. As an additional benefit, the absence of latches also translates into savings of chip area and power consumption.

Clocked-logic, also known as C^{2}MOS, is a particularly attractive approach to gate-level pipelining [2]. In this scheme, the latches are eliminated altogether; instead, a clock is applied directly to the logic gate, to which latching functionality is added.

Fig. 5 shows the structure of a general C^{2}MOS gate. The clock input En directly controls the gate through two transistors, one each in the pull-up and the pull-down network. When En is asserted, the gate is enabled and a new output is produced. When En is deasserted, the gate holds its output value. Typically, an inverter pair providing weak feedback is attached at the gate output to provide a more robust hold operation. While C^{2}MOS has been proposed as a synchronous technique [2], it can be naturally adapted to very high-speed asynchronous pipelines using local handshake signals to replace the clock.

Fig. 6 shows a C^{2}MOS implementation of the MOUSETRAP pipeline. The explicit data latches of Fig. 4 have been eliminated; instead, C^{2}MOS gates provide both logic as well as latching functionality. Note that the "clock" input to the C^{2}MOS logic is actually the asynchronous En signal that is locally generated by the XNOR gate in the stage's latch controller. Both En and $\overline{En}$ are needed for the control of C^{2}MOS gates. This suggests the use of a *dual-rail* XNOR gate, which is discussed next.

3) XNOR *Optimization: Dual-Rail Implementation:* An alternative design of the pipeline control is now proposed, to eliminate two gate delays from the critical path. Since many transparent latches as well as C^{2}MOS gates require both true and complemented enables, a useful optimization for both of the proposed pipeline schemes (Figs. 4 and 6) is to implement the XNOR as a *dual-rail* gate, providing both XOR and XNOR outputs. As highlighted in Fig. 6, the XNOR now has two dual-rail inputs—(done, $\overline{\text{done}}$) and (ack, $\overline{\text{ack}}$)—and a dual-rail output (En, $\overline{En}$). Accordingly, the "bit latch"—which receives the incoming req, and generates done as well as the outgoing req and ack signals—is now replaced by a pair of C^{2}MOS identity gates. While this approach increases the overall control area, it directly improves the performance: two inverters are eliminated from the critical cycle (from XNOR inputs and its output).

D. Pipeline Performance and Timing Constraints

This section presents an analytical evaluation of both pipeline performance and timing constraints.

1) Performance: Two key measures of the performance of the pipeline are discussed: *forward latency* and *cycle time* [39].

Forward latency is the time it takes a data item to pass through an initially empty pipeline. Since, in an empty pipeline, all the latches are transparent, the pipeline latency per stage L is simply the stage's latch delay plus logic delay

$$L = t_{\text{Latch}} + t_{\text{logic}}. \tag{1}$$

Cycle time is the time interval between successive data items emerging from the pipeline when the pipeline is operating at maximum speed. A cycle of stage N, from one enabling of its latch to the next, consists of three events: 1) new data passes through the latch and the stage's logic block, 2) the data passes through stage $N+1$'s latch, producing ack_N, and 3) ack_N causes stage N's latch controller to reenable stage N's latch. Therefore, the analytical cycle time T is

$$T = t_{\text{Latch}} + t_{\text{logic}} + t_{\text{Latch}} + t_{\text{XNOR}\uparrow} \tag{2}$$
$$= 2 \cdot t_{\text{Latch}} + t_{\text{logic}} + t_{\text{XNOR}\uparrow} \tag{3}$$

where t_{logic} is the delay through the logic block, and $t_{\text{XNOR}\uparrow}$ is the time it takes the XNOR gate to enable the latch.

For the special case of C^2MOS pipelines, there are no explicit latches. If the delay through a C^2MOS gate is denoted by $t_{\text{C}^2\text{MOS}}$, the latency and cycle time are given by

$$L_{\text{C}^2\text{MOS}} = t_{\text{C}^2\text{MOS}} \tag{4}$$
$$T_{\text{C}^2\text{MOS}} = 2 \cdot t_{\text{C}^2\text{MOS}} + t_{\text{XNOR}\uparrow}. \tag{5}$$

The cycle times of (3) and (5) are quite fast, and would be difficult to surpass with synchronous schemes. For example, a standard synchronous pipeline, with alternating latches controlled by complementary clocks, and with logic between every adjacent latch pair, will typically have a cycle time of at least $2 \cdot t_{\text{Latch}} + 2 \cdot t_{\text{logic}}$, plus adequate margins to compensate for clock skew and jitter.

2) Timing Constraints: There are two simple one-sided timing constraints which must be satisfied for the correct operation of the pipeline: *setup time* and *data overrun*. (A third one-sided constraint, which is standard in such asynchronous design styles, is the "bundling constraint" (Section II-A), which requires that the latency through the matched delay in each stage is greater than the worst-case logic delay.)

Setup Time. Once a latch is enabled and receives new data at its inputs (along with a req signal), it must remain transparent long enough for data to pass through. Thus, the path from req_N to En deasserted (XNOR switching low) must be longer than the setup time t_{setup}

$$t_{\text{req}_N \rightarrow \text{done}_N} + t_{\text{XNOR}_N\downarrow} > t_{\text{setup}}. \tag{6}$$

This constraint is usually easily satisfied because the delay from req_N to done_N typically exceeds the setup time.

Data Overrun (Hold Time): Once data enters a stage, it should be securely captured before new data is produced by the previous stage. If this condition is violated, stage N's data will be overwritten by new data. Therefore, since ack_{N-1} and done_N are generated in parallel, the path from ack_{N-1} to stage N's data inputs must be longer than the time to close N's latch, plus a hold time t_{hold}

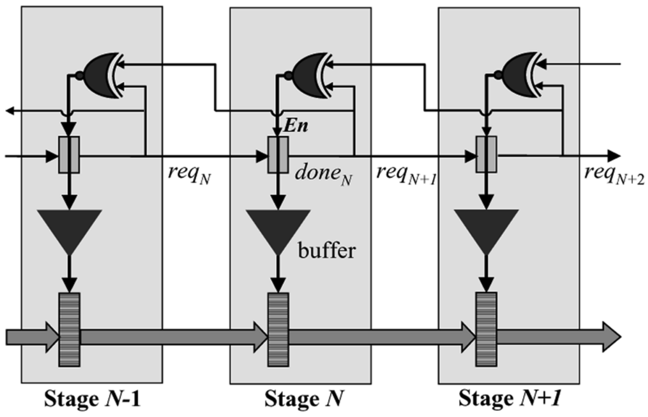

Fig. 7. Handling wide datapaths in MOUSETRAP pipelines.

$$t_{\text{XNOR}_{N-1}\uparrow} + t_{\text{Latch}_{N-1}} + t_{\text{logic}_{N-1}} > t_{\text{XNOR}_N\downarrow} + t_{\text{hold}}. \tag{7}$$

The left terms represent the shortest path through the XNOR to the arrival of new input from stage $N-1$. The right terms represent the path to disabling stage N's latch. The equation can be rewritten to simplify the constraint

$$t_{\text{Latch}_{N-1}} + t_{\text{logic}_{N-1}} > (t_{\text{XNOR}_N\downarrow} - t_{\text{XNOR}_{N-1}\uparrow}) + t_{\text{hold}}. \tag{8}$$

Assuming $t_{\text{XNOR}_{N-1}\uparrow} \cong t_{\text{XNOR}_N\downarrow}$, the right expression in parentheses is canceled. The result is a simple hold time constraint, which is usually easily satisfied because the latch and logic delays through stage $N-1$ typically exceed the hold time.

E. Handling Wide Datapaths

An important practical issue in designing asynchronous pipelines is the handling of very wide datapaths, where a single control signal for a pipeline stage must be broadcast across many latches. In principle, such control distribution may introduce sizable delays in the critical path, slowing down the operation of the pipeline. There are two practical solutions proposed for efficient pipelining of wide datapaths: 1) *datapath partitioning* and 2) *control kiting*.

In the first approach, *datapath partitioning*, a wide datapath is divided into several smaller independent streams. The pipeline control is replicated for each stream, and each stream has its own sequence of completion generators and matched delays. As a result, the requirement of buffering is significantly reduced: in each stage, the latch controller generates a latch enable signal which is broadcast to only a small number of bits, i.e., to only those bits that lie inside that partition. This approach is typically applicable to bit-slice datapaths, such as plain FIFOs and logic function units (e.g., bitwise AND, OR, etc.).

The second approach to handling wide datapaths, called *control kiting*, allows the datapath to be skewed with respect to the control [21], [42]. No partitioning is used; instead, buffers are inserted to adequately amplify the latch enable signals which drive the datapath latches. However, the latch enables for the completion generators do not need this amplification; they are simply tapped off from *before* the buffers, i.e., the control path

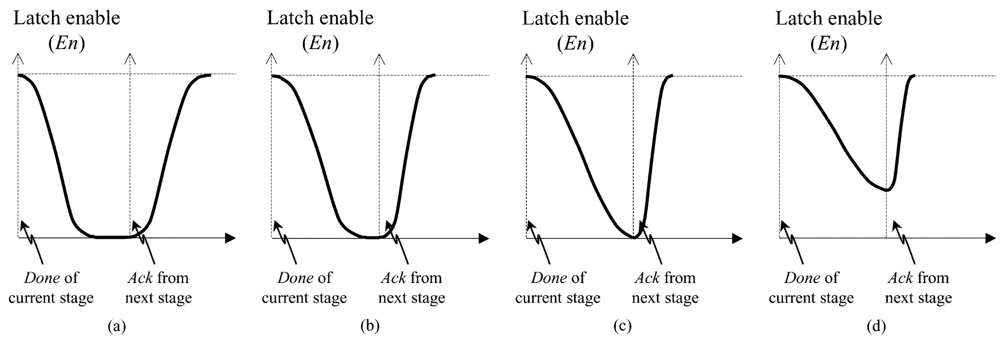

Fig. 8. Waveform shaping optimization. (a) No waveform shaping: symmetric rise and full times. (b) Waveform shaping: asymmetric rise and fall times. (c) Limiting case: little timing margin. (d) Extreme case: reduced voltage swing.

does not incur the buffering delays used in the datapath, and is therefore *skewed* with respect to the datapath. As a result, much of the overhead of broadcasting the latch enable to the datapath is hidden, occurring in parallel with other pipeline operations.

Fig. 7 shows an example of how control kiting can be implemented in a MOUSETRAP FIFO. There are two key differences with respect to the basic implementation of Fig. 1. First, the "bit latch," which receives the incoming req and generates the outgoing req and ack signals, is now pulled apart from the rest of the latch which handles the data bits. Second, buffers are added in each stage to amplify the enable signal that controls the lower part of the latch; the "bit latch" still receives the unamplified enable. Since the insertion of buffers only delays the latching (and unlatching) of the datapath, the completion signal of each stage, req, is actually produced a buffer delay earlier than the data outputs.

Interestingly, if the buffer delays are assumed to be uniform, the pipeline of Fig. 7 not only operates correctly, but also has *exactly* the same analytical cycle time and timing constraints as those derived for narrower datapaths. In particular, the timing analysis of Section II-D still applies to the controller circuits; the only difference is that the datapath is now operating at a constant skew with respect to the controllers. However, if buffer delays are unequal, the analysis needs to be modified to account for this difference. Assuming that the difference between buffer delays in neighboring stages is bounded by $t_{\text{buff-skew}}$, the timing constraints of (6) and (8) are rewritten as

$$t_{\text{req}_N \to \text{done}_N} + t_{\text{XNOR}_N\downarrow} > t_{\text{setup}} + t_{\text{buff-skew}} \tag{9}$$

$$t_{\text{Latch}_{N-1}} + t_{\text{logic}_{N-1}} > (t_{\text{XNOR}_N\downarrow} - t_{\text{XNOR}_{N-1}\uparrow}) + t_{\text{hold}} + t_{\text{buff-skew}}. \tag{10}$$

Each of the two approaches to handling wide datapaths has acceptable overheads. In particular, the first approach, datapath partitioning, may sometimes introduce forks and joins into the datapath if the individual data streams are not entirely independent. These forks and joins can be handled through a slight modification to pipeline control, as shown in Section II-G, incurring only a modest overhead to pipeline performance. The second approach, control kiting, also has an overhead: it requires the buffering delays to be fairly uniform across pipeline stages; otherwise, the timing margins available to satisfy the constraints of Section II-D2 are effectively reduced. The practicality of control kiting, however, has been demonstrated through a fabricated FIR filter chip developed jointly with IBM, although using a different pipeline style [32], [36]. The filter exhibits a highly-varied datapath, ranging from 30 to 216 wires in width at different stages in the pipeline, yet was successfully handled using control kiting.

F. Pipeline Speedup: Optimized Control Generation by "Shaping" XNOR *Output*

This subsection presents a circuit-level optimization that can further improve the pipeline's performance *under steady-state operation*. The key idea is to shape the output of the latch controllers through transistor sizing, such that the critical cycle is further shortened at the expense of some loss of timing margins. By varying the aggressiveness of this optimization, the designer can generate a whole set of implementations, ranging from a robust unoptimized implementation to an aggressive implementation partially similar to a *wave pipeline*, but with greater robustness [15], [19], [40].

In particular, in wave pipelining, multiple waves of data are propagated between two clocked latches; to ensure data integrity, all path delays between the latches are required to be extremely accurately balanced, and both left and right environments must operate in perfect synchronism. However, unlike wave pipelining, all MOUSETRAP implementations even with the proposed optimization exhibit full asynchronous handshaking, thereby allowing variable environment rates as well as stalls to be gracefully handled. That is, all the MOUSETRAP variants allow *back pressure* through synchronization to enforce stalling when there is congestion; in contrast, wave pipelining typically allows no backwards pressure and will malfunction if there is congestion.

This wave-shaping optimization is implemented by making the rising and falling transitions of the latch controller asymmetric, such that the speed-critical up-transition is made faster at the expense of the down-transition. In particular, the rising transition of the controller is sped up by appropriately sizing the transistors of the XNOR gate; the shorter $t_{\text{XNOR}\uparrow}$ implies improved cycle times [cf. (3) and (5)]. However, to maintain the same amount of loading in the controller circuit, the down-transition is slowed down; the longer $t_{\text{XNOR}\downarrow}$ implies somewhat tighter timing [cf. (8)].

1) Waveform Shaping Scenarios: Fig. 8 shows four different scenarios, which result when the waveform shaping optimization is applied increasingly aggressively from left to right. Each graph represents one complete cycle of operation, from one latch enable to the next latch enable. Moving left to right, the cycle time is shortened, thereby improving throughput, but at the cost of reduced timing margins. Each of the four scenarios is now discussed in detail.

Scenario I (No Waveform Shaping): Fig. 8(a) shows the controller output (i.e., latch enable) when no waveform shaping is applied. The deassertion and the subsequent assertion of the latch enable—i.e., the fall and rise times, respectively—are similar. The throughput obtained is used as a baseline for comparison with the remaining scenarios. The timing margin available to satisfy the *data overrun* timing constraint (8) is, in practice, quite adequate: $t_{\mathrm{Latch}} + t_{\mathrm{logic}} - t_{\mathrm{hold}}$.

Scenario II (Moderate Waveform Shaping): Fig. 8(b) shows the controller output when a moderate amount of waveform shaping is applied. The (re)enabling of the latch (i.e., the up transition) is sped up through transistor sizing. At the same time, the disabling of the latch is slowed down, in order to keep the total load represented by the latch controller (i.e., the XNOR's input capacitance) constant; thus, the delays in the rest of the critical path remain largely unchanged. The net impact is that the cycle time is reduced because the critical up-transition happens faster [see (3) and (5)]. However, the timing margin available to satisfy the data overrun constraint of (8) is somewhat reduced, i.e., by the amount of the asymmetry in the fall and rise times ($t_{\mathrm{XNOR}\downarrow} - t_{\mathrm{XNOR}\uparrow}$). In practice, though, experiments indicate that this constraint can still be usually satisfied easily (see Section V).

Scenario III (Limiting Case): Fig. 8(c) represents the limiting case in which waveform shaping is applied more aggressively, while still barely ensuring both of the following: 1) the controller output has a full voltage swing and 2) the data overrun timing constraint is satisfied, possibly with little margin. This scenario represents the lower bound on the cycle time that can be achieved using waveform shaping, while still barely maintaining correct operation.

Scenario IV (Extreme Case): If waveform shaping is applied beyond the limiting case, the controller output no longer exhibits a full voltage swing, as shown in Fig. 8(d). In this scenario, *under steady-state operation*, the pipeline latches are never fully disabled. As a result, the reenabling of the latches occurs even faster, thereby further shortening the cycle time and improving the throughput. However, the timing constraint of (8) may no longer be satisfied, thereby potentially allowing for the possibility of data overrun unless careful management of datapath delays is undertaken.

The reduced voltage swing scenario has some similarities with wave pipelining, but also some fundamental differences. In particular, if the voltage swing is reduced to the extent that the latches are always substantially transparent, then the entire pipeline operates in a manner similar to flow-through combinational logic. As a result, the throughput obtained will be competitive to that of a wave pipeline. However, reduced-swing MOUSETRAP pipelines are fundamentally more robust because they require much simpler timing assumptions, and can robustly interface with variable-speed environments. In particular, even though their latches do not close fully under steady-state operation, they are still fully functional. In the event of a slowing down of the receiving environment, or due to stalls in the pipeline, the latches will fully close to secure the data they are holding, thereby gracefully handling situations that wave pipeline cannot handle. An in-depth comparison with wave pipelines is provided in Section IV-C.

2) Revisiting Timing Constraints: Handling Waveform Shaping Optimization: Since the proposed waveform shaping optimization somewhat worsens the timing margin available to satisfy the data overrun timing constraint, that constraint is now revisited and analyzed in greater detail. This constraint must be satisfied in order for the latches to retain their full capture funtionality.

In particular, the timing constraint of (8) is further refined by substituting each term, as appropriate, with its minimum or maximum value. Thus, the *slowest* time to disable the latch of stage N must be shorter than the *fastest* time for stage $N-1$ to react to an acknowledgment from N and produce a new data item at the inputs to stage N

$$t^{\min}_{\mathrm{Latch}_{N-1}} + t^{\min}_{\mathrm{logic}_{N-1}} > \left(t^{\max}_{\mathrm{XNOR}_{N}\downarrow} - t^{\min}_{\mathrm{XNOR}_{N-1}\uparrow}\right) + t^{\max}_{\mathrm{hold}}. \tag{11}$$

In the absence of processing logic, this timing constraint requires latch delays to be greater than the amount of asymmetry introduced in the rise and fall times of the latch enables by the waveform shaping strategy, plus a hold time. In practice, this constraint is typically satisfied fairly easily, as indicated by the simulation results in Section V. However, in the event that the timing margin available to satisfy this constraint is insufficient, additional processing logic can be inserted in pipeline stages, which makes the constraint more easily satisfiable.

3) Summary: In summary, the new waveform shaping optimization can help increase throughput at a modest cost in the timing margins. The cost can be negligible if processing logic is added to the pipeline, but is often reasonable even for plain FIFOs. With the optimization, the performance of the pipeline approaches that of wave pipelining, but without the accompanying challenge of aggressively balancing path delays. Further, the designer can choose the aggressiveness of this optimization, and accordingly generate a continuum of implementations ranging from robust unoptimized versions to those that are similar to wave pipelines.

G. Nonlinear Pipelining

The paper has so far focused on linear pipelines, which have many practical applications. However, in complex system architectures, nonlinear pipelining is often needed (Fig. 9). Two simple primitives—fork and join—are now introduced to extend the applicability of MOUSETRAP pipelines.

Figs. 10 and 11 show simple structures for fork and join components, respectively. In a fork stage, the data output and corresponding "req" (matched done output) are both simply forked to the two or more destination stages. In turn, the two or more "ack" signals are combined through a Müller *C-element*. A C-element is an "event ANDer": its output makes a transition when all of its inputs change exactly once [35].

Analogously, in a join stage, the "ack" is simply a forked wire communicating with all sender stages, but there are multiple

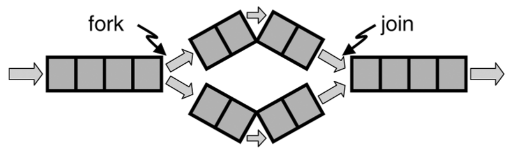

Fig. 9. Nonlinear pipelining.

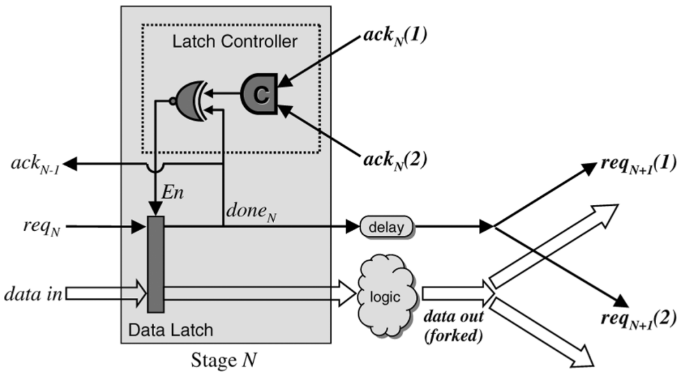

Fig. 10. MOUSETRAP fork stage.

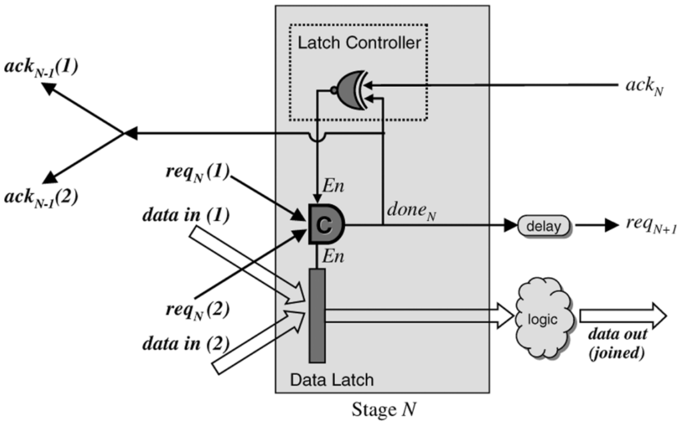

Fig. 11. MOUSETRAP join stage.

"req's" that must be combined. Once again a C-element can simply be used to combine the multiple requests, and the result treated as a unified request that is fed into the latch.

However, an alternative join implementation merges this C-element (that combines multiple requests into one) with the latch itself, thereby producing a single component for greater area efficiency and better latency. This combination of a C-element and a latch is actually an "asymmetric" C-element [11]. Whenever the "latch enable" is asserted, the component's output is 1 when all of the merged "req's" are 1, and is 0 when all of the merged "req's" are 0. At all other times (when the "latch enable" is deasserted, or if the req's are not all equal), the component simply holds its value. At the transistor level, the pull-down network is a single series stack with one transistor for each req, as well as a transistor for the "latch enable." Similarly, the pull-up network is a single series stack with one transistor for each req, and with a transistor for the complemented "latch enable." Finally, the multiple data input streams are simply merged into one stream and latched together.

The analytical cycle time of fork and join structures will be slightly higher than that of linear pipelines, because of the introduction of the C-elements. In particular, for a fork stage (Fig. 10), the cycle time increases from that in (3) by an amount equal to the latency of the C-element (t_C)

$$T_{\text{fork}} = t_{\text{Latch}} + t_{\text{logic}} + t_{\text{Latch}} + t_C + t_{\text{XNOR}\uparrow} \quad (12)$$
$$= 2 \cdot t_{\text{Latch}} + t_{\text{logic}} + t_C + t_{\text{XNOR}\uparrow}. \quad (13)$$

For a join stage (Fig. 11), however, the cycle time does not change much, since the new asymmetric C-element *replaces* the bit latch and has a similar latency (taC)

$$T_{\text{join}} = t_{\text{aC}} + t_{\text{logic}} + t_{\text{Latch}} + t_{\text{XNOR}\uparrow}. \quad (14)$$

In summary, in practical terms, only forks increase the analytical cycle time, and that too only by the amount t_C, which typically represents a small overhead. As an example, if t_{Latch}, t_C, and t_{XNOR} are all assumed to be two gate delays, and if the logic in the stage is also two gates deep, then the cycle time in the presence of a fork (13) will be ten gate delays, instead of eight gate delays for a linear pipeline (3), representing an overhead that is usually quite acceptable. However, we are currently developing optimized versions which reduce this overhead further through careful use of logic decomposition.

H. Pipeline Initialization

Initialization is an important aspect of any design. For simplicity of discussion, this issue has been ignored so far, but is now briefly addressed.

Initialization is achieved very simply by adding a *global "reset"* input to the latch of every pipeline stage, thereby making it resettable. In particular, an extra pull-up transistor controlled by *reset* is added to the latch, which pulls the latch's internal node high during initialization, thereby causing the latch output to reset low. Therefore, all the done, req, and ack signals are forced low during initialization, which in turn asserts all the latch enables (En). As a result, all pipeline stages are initialized to be empty, and all latches are initialized to be transparent while they await the first data item. Once the pipeline is thus initialized, *reset* is deasserted. The pipeline is then ready for operation.

The initialization capability is implemented with a low overhead. In practice, only that one bit of the latch which produces the done signal requires this reset capability. For the dual-rail implementation of Section II-C3, the pair of identity C^2MOS gates that produce the dual-rail done output are initialized so that the true rail is low and the complement rail is high. The pipeline circuits designed and simulated for this paper all include full initialization capability.

III. Overview of Related Work

A. Synchronous Pipelines

Several synchronous pipelines have been proposed for high-throughput applications. In *wave pipelining*, multiple waves of data are propagated between two latches [19], [40]. However, this approach requires much design effort, from the architectural level down to the layout level, for accurate balancing of path delays (including data-dependent delays), and remains highly vulnerable to process, temperature, and voltage variations. Section IV-C provides a detailed comparison between wave pipelining and MOUSETRAP.

Other aggressive synchronous approaches include *clock-delayed domino* [41], *skew-tolerant domino* [14], and *self-resetting circuits* [23], [8]. These all require complex timing constraints which are difficult to verify; they also lack elasticity and still require high-speed global clock distribution.

Several recent approaches combine clocking with the benefits of handshaking, and thereby obtain *elasticity* or *latency insensitivity* within a synchronous system. Carloni *et al.* [3] introduced a formal approach for latency-insensitive design of single-clock systems. Synchronous IP blocks are encapsulated inside custom wrapper circuits, which employ clock gating when necessary to accommodate arbitrary communication latencies. The approach of Jacobson *et al.* [16] pushes latency-insensitive design to the granularity of individual latches. Recently, Singh and Theobald [31] have introduced several extensions to the notion of latency insensitivity, including generalization from single-clock to multiclock systems, and from point-to-point communication to arbitrary network topologies.

Finally, several approaches have been proposed recently for implementing communication across clock boundaries in multiclock systems. Chelcea and Nowick [5] introduced low-latency asynchronous FIFOs that can glue together different timing domains. An alternative approach by Chakraborty and Greenstreet [4] features a novel technique for handling metastability, which practically eliminates the performance overheads of other approaches (e.g., a double-latch approach). Finally, Kessels *et al.* [17] build upon the pipeline design of this paper and introduce a clocked version of MOUSETRAP for bridging together different clock domains. MOUSETRAP was chosen because its two-phase protocol had the advantage of fewer round-trip delays (compared with four-phase styles), and because this pipeline could be easily implemented using any standard cell library.

B. Asynchronous Pipelines

There has been much work recently on asynchronous pipelines. This subsection presents a survey of recent asynchronous pipeline styles, first single-rail and then dual-rail.

1) Single—Rail: The classic single-rail asynchronous pipelines introduced by Sutherland are called *micropipelines* [35]. This style uses elegant control, but has slow and complex capture-pass latches which can significantly hinder performance. A number of variants using alternative control and latch structures have been proposed [7], [21], [42] but in each case the performance is limited due either to excessive control delays or to sizable latch delays. Recently, a new style, GasP, has been proposed which obtains even higher throughputs [33]. However, this approach requires fine-grain transistor sizing to achieve near-exact delay equalization for all gates in the control circuitry, and the protocol has more complex two-sided timing constraints. In contrast, MOUSETRAP pipelines do not require delay equalization and have simpler one-sided timing constraints.

The fastest designs reported so far are the IPCMOS pipeline from IBM [25] and the GasP pipeline from Sun. Both of these approaches are targeted to bundled datapaths [26] that use static logic, i.e., datapath blocks with an attached worst-case delay line (discussed in Section II-A). Each style provides very high throughputs through use of novel complex control structures and aggressive circuit techniques. Throughputs of 3.3 GHz in 0.18 μm were reported for IPCMOS in an IBM proprietary silicon-on-insulator technology, at the normal process corner.[2] In contrast, the experimental results for MOUSETRAP are for a standard bulk-CMOS process, which is very likely significantly more conservative than the proprietary IBM process used for IPCMOS. For GasP, a throughput of 1.5 GHz has been reported in 0.35-μm technology.

While a definitive quantitative comparison of MOUSETRAP with IPCMOS and GasP is not possible due to the significant differences in the fabrication technology, a reasonable technology-independent comparison can be made by evaluating the cycle times of these approaches in terms of a number of CMOS logic levels on the critical path. In particular, a MOUSETRAP FIFO has only 5–6 levels of CMOS gates plus one pass gate (two latches and one XNOR) on its critical cycle. While GasP also has a similar critical cycle (6 levels of CMOS logic), the IPCMOS cycle is much longer: 12 levels of CMOS logic plus one pass gate. Thus, if implemented in the same technology, MOUSETRAP is expected to outperform IPCMOS and have a performance similar to that of GasP. Arguably, the gates used to implement GasP may have a somewhat lower logical effort [34] compared with MOUSETRAP, but there are other features of GasP (e.g., significantly more complex timing requirements and transistor sizing) which make MOUSETRAP an attractive alternative.

In summary, while IPCMOS and GasP are very fast, they are much more complex and designer-intensive styles. MOUSETRAP provides a different point in the design space: much greater simplicity, with nearly comparable (or better) performance. Section IV-B provides a detailed technology-independent comparison.

2) Dual-Rail: The classic dual-rail asynchronous pipeline approach is the PS0 pipeline by Williams and Horowitz [39]. This pipeline style uses dynamic logic for the datapath and uses dual-rail encoding with completion detection to generate control information. A key contribution of this approach is elimination of explicit latches or storage elements from the datapath; latching functionality is achieved instead by a careful sequencing of control of the dynamic function blocks.

There have been a number of recent approaches that build upon the PS0 pipeline and achieve significantly higher performance [27]–[29], [32]. In particular, throughputs up to 2.4 GHz in 0.18-μm technology have been reported for FIFOs, and up to 1.3 GHz in the same technology for a complex real-world digital FIR filter chip [32].

In addition, dynamic pipelines have been proposed that aim at improving the storage efficiency of the pipeline. In particular, PS0 is limited to storing distinct data items only in alternating

[2]Throughputs of up to 4.5 GHz have been reported, but these are only for extreme process cases ($L_{\text{eff}} = -2\sigma$ and low V_t).

pipeline stages, thus offering at most a 50% storage capacity. In contrast, the high-capacity (HC) pipeline of [28] and [32] offers 100% storage capacity as well as higher throughput than PS0.

Martin *et al.* [20] and Lines [18] present the design of a complete microprocessor using fine-grain pipelining techniques similar to Williams'. The pipeline circuits are based on the conservative and robust quasi-delay-insensitive (QDI) model, yet have high performance. While several distinct pipeline styles are introduced, the most commonly-used ones have cycle times that are 18 gate delays or greater. Other QDI styles include those by Ozdag and Beerel [24], which improve upon the performance of [18] and [20] through use of concurrency reduction to simplify the design of the dynamic function blocks (i.e., reduced stack size). Finally, Ferretti and Beerel [9] combine the robustness of delay-insensitive encoding with the effiency of the single-track handshaking of [33] and introduce single-track 1-of-N pipelines.

While dual-rail asynchronous pipelines typically provide greater robustness than single-rail pipelines, the former are less compatible with current industry design methodologies. In particular, many dual-rail approaches use dynamic logic, which currently enjoys much less commercial tool support as compared to static logic. Dynamic logic also has the inherent problem of vulnerability to noise, thereby requiring careful custom design to ensure reliable operation. Finally, dual-rail design typically requires the creation of new function blocks, since it cannot easily reuse preexisting designs which are predominantly single rail. As a result, single-rail bundled datapath approaches have the attractiveness of greater compatibility with industry practices.

IV. Detailed Comparison With Related Work

The proposed MOUSETRAP pipeline is now compared in greater detail with several recent closely-related approaches. First, Section IV-A provides a comparison with other classic transition-signaling pipelines. Then, Section IV-B compares MOUSETRAP with two of the fastest existing pipeline styles: GasP pipelines from Sun and IPCMOS pipelines from IBM. Finally, a detailed comparison with wave pipelining is provided in Section IV-C.

A. Comparison With Other Transition-Signaling Asynchronous Pipelines

Previous transition-signaling asynchronous pipelines fall into two categories: those that use phase conversion [7], [22], [35], and those that do not [11], [42].

The pipelines of ([35, Fig. 14]) and ([7, Fig. 10]), called *micropipelines*, both use phase conversion. Similar to MOUSETRAP, a micropipeline stage uses transition signaling and transparent latches (see Fig. 12). However, both of these approaches have significantly more complex control than MOUSETRAP. Each has two extra components per stage: a C-element and a Toggle element. The Toggle element routes transitions received on its input to one of two outputs alternately, starting with the output labeled with a dot. The critical paths are also much longer than MOUSETRAP: from req_N to req_{N+1} there are four component delays (the C-element, XNOR, latch, and Toggle), and

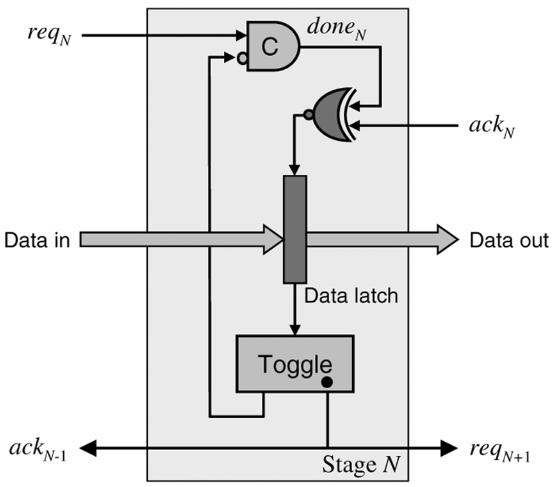

Fig. 12. Sutherland's and Day/Woods' micropipeline stage.

from ack_N to the input of the C-element (to half-enable it) there are three component delays (the XNOR, latch, and Toggle).

By comparison, MOUSETRAP pipelines have significantly simpler structure than the micropipelines of Fig. 12. In particular, the critical paths are much shorter: there is only a single latch delay from req_N to req_{N+1}, and only an XNOR delay from ack_N to the reenabling of the stage for the next data item. In fact, as seen in the alternate view of MOUSETRAP of Fig. 2, the top bit of the latch (labeled "bit latch") performs the combined role of both the C-element and the Toggle element of Fig. 12, i.e., it receives the incoming request and generates the outgoing request and acknowledgment.

This difference in structure is critical: it allows not only faster operation for MOUSETRAP, but also allows systems built using MOUSETRAP pipelines to be implemented using standard components (latches and XNORs). In contrast, the micropipelines of Fig. 12 require C-elements and Toggle elements, which typically must be custom designed as they are not supported by standard cell libraries or by automated design and test tools.

Closer to MOUSETRAP, the Sun Laboratories' "Charlie boxes" [22] include simpler designs, such as the so-called *S-style*. MOUSETRAP pipelines can be regarded as more optimized—and less robust—versions of some Charlie boxes. For instance, MOUSETRAP pipelines generate an earlier completion signal, thereby obtaining shorter cycle times at the modest expense of a timing assumption that is in practice usually easy to satisfy [cf. (8)]. In addition, MOUSETRAP provides new templates for handling nonlinear datapaths (i.e., forks and joins). Furthermore, several novel optimizations are introduced for MOUSETRAP—a "waveform shaping" strategy to speed up the critical path; an inverter elimination strategy using dual-rail control logic; and the use of a C^2MOS-style—none of which appeared in [22].

There are several alternative approaches to implementing phase conversion, between transition-signaling and four-phase protocols. However, each of these approaches has greater complexity and analytical cycle time. In [11], Furber and Day propose three distinct four-phase protocols for asynchronous pipelines: fully decoupled, long hold and semidecoupled. In the first two, pipeline control is significantly more complex than in MOUSETRAP. The best of their designs, semidecoupled, introduces a highly-concurrent protocol, but still has a minimum of four components on the critical cycle. These components are all C-elements, two of which have stack depth of 3, and additional inverters are actually implied for correcting polarity. In contrast, MOUSETRAP only has three components on the critical cycle (two transparent latches and an XNOR), no stack depths of 3, no implied inverters, and avoids the extra switching activity of four-phase communication.

Compared with these phase-conversion approaches, MOUSETRAP can be regarded as more of an optimized variant, which, in some cases, may be slightly less robust. In particular, the acknowledge signal in MOUSETRAP is sent to the previous stage as soon as data has been received, but before it has been securely latched in the current stage. This optimization significantly improves the pipeline performance, but introduces a small hold time requirement [see (8)]. This timing constraint is typically easily satisfied in practice, as demonstrated by simulation results, and therefore justifies the optimization. The other approaches delay acknowledgment until after the data has been latched. As a result, these approaches meet the hold time constraint by construction, but at the cost of lower performance.

A final alternative approach to phase conversion is to retain transition-signaling control, but replace the transparent latches with dual-edge-triggered D-flip-flops (DETDFF's) [42]: data is latched each time the latch control is toggled. While this approach avoids the overhead of phase conversion, it incurs a heavy performance penalty because DETDFF's are significantly slower than transparent latches, and are also much larger.

B. Comparison With GasP and IPCMOS Pipelines

There are three fundamental distinctions between MOUSETRAP and the most competitive recent high-speed asynchronous pipelines from Sun Laboratories (GasP [33]) and IBM (IPCMOS [25]). Overall, MOUSETRAP has the significant advantage of requiring less designer effort: it has fewer requirements on transistor sizing, uses more standard VLSI circuit structures, and has simpler timing constraints. Each of these three distinctions are now discussed in detail.

First, the design methodologies of GasP and IPCMOS inherently expect very *careful transistor sizing*. In particular, the GasP approach aims for fine-grain transistor sizing to make *all gate delays equal*. As a result, it can be argued that the GasP circuits are not really asynchronous at the level of individual gates, but are completely balanced and timed: the cycle time is defined by a fixed number of equal gate delays on the critical path. The pipeline's asynchronous behavior is observed only at a higher level of abstraction, i.e., at the level of pipeline stages. In IPCMOS, inter-stage communication is performed using pulses instead of level signals. As a result, careful sizing of output drivers is needed to ensure integrity of pulsed signals. In contrast, while MOUSETRAP benefits from careful transistor sizing, it is not a fundamental requirement for correct operation. Furthermore, MOUSETRAP uses transition signaling, thus avoiding the challenging signal integrity issues of IPCMOS's pulse-mode communication.

Second, both the GasP and IPCMOS styles require *special-purpose VLSI circuit structures*. In particular, GasP uses aggressive nonstandard techniques such as communicating control information on tristated wires, bidirectional communication on the same wire, and the use of self-resetting gates in the control logic. Similarly, IPCMOS uses pseudo-nMOS structures that may experience short-circuit conditions, thereby requiring careful design to avoid malfunction. In addition, IPCMOS uses switches based on pass-transistor logic. In contrast, MOUSETRAP pipelines can be implemented using standard cells if needed: standard transparent latches and static logic for both datapath and control, without requiring any specialized gate styles or components.

Finally, the timing constraints in GasP and IPCMOS are significantly more stringent than the requirements of MOUSETRAP. In particular, GasP has *two-sided timing constraints* for correct operation. The pulse generated inside the control of each pipeline stage (when new data arrives) must be long enough in order to effectively charge/discharge both the left and right state conductors. However, that pulse must also be short enough so that it does not overlap with the pulses generated inside neighboring stages; otherwise, the pulses in each pair of adjacent stages will "fight" for control of the state conductor, causing periods of short-circuit activity with unpredictable behavior. Thus, there is a two-sided constraint on the width of pulses generated by each GasP control circuit. IPCMOS has similar two-sided constraints on the widths of the pulses used for inter-stage communication. In contrast, MOUSETRAP only has simple setup and hold time requirements, which are single-sided and, in practice, usually easy-to-satisfy.

C. Comparison With Wave Pipelining

The steady-state performance of MOUSETRAP is competitive with that of a *wave pipeline* [15], [19], [40]. In particular, in a wave pipeline, multiple waves of data are propagated between a pair of latches; there are no other latches in the datapath. By comparison, under steady-state operation, MOUSETRAP also acts as a transparent flow-through combinational logic. In particular, when the waveform shaping optimization of Section II-F is used, it is possible to keep the latches continuously transparent, when a steady stream of data is being processed. Hence, under steady state, the performance of MOUSETRAP should be competitive with that of wave pipelines. At the same time, much like in a wave pipeline, careful management of datapath delays will be required in order to preserve data integrity. Otherwise, for example, certain bits of data in a stage may be overrun by new data bits arriving along "fast paths" from its preceding stage.

However, reduced-swing MOUSETRAP pipelines are fundamentally more robust than wave pipelines. There are two key differences. First, careful delay management is less critical for MOUSETRAP than for wave pipelining. The reason is that, in steady-state operation, even if the latches are not made fully opaque, they still go through a phase in which they are not fully transparent either. As a result, the transmission of data bits is somewhat slowed momentarily (though not fully stopped)—especially for the new data bits arriving along "fast

TABLE I
PERFORMANCE OF MOUSETRAP FIFO (0.18-μm TSMC TECHNOLOGY)

Pipeline Design	latch delay t_{Latch} (ps)	XNOR delay $t_{\text{xnor}\uparrow}$ (ps)	$t_{\text{xnor}\downarrow}$ (ps)	Cycle Time, T Analytical Formula	(ps)	Throughput (GigaHertz)
MOUSETRAP	188	102	115	$2 \cdot t_{\text{Latch}} + t_{\text{xnor}\uparrow}$	477	2.10
MOUSETRAP$_{opt}$	179	63	131	$2 \cdot t_{\text{Latch}} + t_{\text{xnor}\uparrow}$	421	2.38

paths"—thereby somewhat mitigating the task of careful management of datapath delays. Thus, the presence of handshake signals in MOUSETRAP allows some degree of synchronization even in the reduced voltage swing scenario. In contrast, wave pipelining requires a highly accurate balancing of path delays (including data-dependent delays), and remains highly vulnerable to process, temperature, and voltage variations.

Second, and more importantly, MOUSETRAP provides much greater robustness than wave pipelining in the presence of pipeline *congestion* or environment *stalls.* In contrast, wave pipelining fundamentally cannot handle these nonsteady-state situations. For instance, consider the interface of a MOUSETRAP pipeline with its right environment. Whether the right environment suddenly stalls or speeds up, the pipeline gracefully handles these variations. If the right environment is slow and cannot respond with an ack, the rightmost pipeline stage quickly makes its latch opaque (since no ack is received by its XNOR), thus preventing an overrun from the left stage. Even if the right environment is very fast, it is correctly stalled until the rightmost stage can deliver it data, since the environment is waiting for the stage's req signal. The same reasoning also applies to the internal stages in the pipeline, making the pipeline robust to internal delay variations and congestion as well. In contrast, a wave pipeline lacks handshaking, and therefore cannot adapt to variations in input or output rates or to pipeline congestion and environment stalls.

V. EXPERIMENTAL RESULTS

Results of post-layout SPICE simulation for basic MOUSETRAP pipelines are now presented.

Experimental Setup: A simple 10-stage FIFO was simulated (with no logic processing). The FIFO was laid out using the Cadence tool suite in a 0.18-μm TSMC process. Two versions of the pipeline were simulated: 1) the "unoptimized" pipeline style, i.e., without the waveform shaping optimization of Section II-E, but with dual-rail control and 2) an "optimized" version with waveform shaping, corresponding to the scenario to Fig. 8(d).

Both a 4-bit FIFO and a 16-bit FIFO were simulated for the unoptimized style. Identical control circuits were used in both the cases (including identical transistor sizes), but control kiting was used for the 16-bit design to handle the wider datapath without any performance degradation (see Section II-F). For the optimized style, the waveform shaping optimization was performed on the 4-bit FIFO to obtain further improvement in throughput, though at the expense of some loss of timing margins. This optimization was not performed, however, on the 16-bit FIFO because control kiting used on that design already meant potentially tighter timing margins.

The operating conditions were 1.8 V nominal voltage supply, 300 K temperature, and a normal process corner. Simple custom cells were designed: a pass-gate implementation of a dual-rail XNOR/XOR pair, and an eight-transistor dynamic D-latch. (However, it should be noted that MOUSETRAP can be easily implemented using any standard cell library as well.)

Simulation Results: Table I summarizes the simulation results. The overall pipeline cycle time T is given, as well as the delays of individual components: the latch delay t_{Latch} and controller gate delays $t_{\text{XNOR}\uparrow}$ and $t_{\text{XNOR}\downarrow}$. The first row, labeled "MOUSETRAP," presents results for *both* the 4- and 16-bit FIFOs without the waveform shaping optimization. Since control kiting effectively isolates the control circuits from the higher load represented by the wider datapath of the 16-bit design, no significant difference was observed in the simulations for the 4- and 16-bit FIFOs. The timing constraints of Section II-D2 were easily satisfied with adequate margins. The second row of the table, labeled "$\text{MOUSETRAP}_{\text{opt}}$," represents the results for the 4-bit design with somewhat aggressive waveform shaping. The throughputs obtained were quite encouraging: 2.1 GHz for the unoptimized design and nearly 2.4 GHz for the optimized one.

These numbers compare favorably to the IPCMOS style of Schuster *et al.* [25]. Their reported results of 3.3 GHz are for a high-performance IBM 0.18-μm process; this IBM process is in practice significantly faster than the 0.18-μm TSMC process we used. In particular, the IBM process used was silicon-on-insulator (SOI), whereas our process was a bulk process. Although our designs do not include logic processing, we anticipate competitive performance with IPCMOS using a comparable process when logic is included, since one gate delay of logic adds little to the overall MOUSETRAP cycle time. As indicated earlier, the IPCMOS critical path is made up of 12 levels of CMOS logic, plus a pass gate. In contrast, our MOUSETRAP implementation only uses five levels of CMOS logic, plus a pass transistor, on its critical path (plus two additional CMOS levels if there are forks and joins in the datapath). In addition, MOUSETRAP has the benefit of much simpler circuit components and timing constraints.

The simulations also demonstrate the benefit of the waveform shaping approach. The table shows that the optimization resulted in an increase in throughput from 2.10 to 2.38 GHz, representing a 13% improvement. The table shows how the optimization led to a decrease in $t_{\text{XNOR}\uparrow}$, which is on the critical path, in exchange for a slightly longer $t_{\text{XNOR}\downarrow}$, which is not on the critical path.

Fig. 13 shows the waveforms corresponding to the results in the table. The top half of the figure shows the done output and the latch enable signal for one pipeline stage of the unoptimized design, i.e., without waveform shaping. The bottom half shows the waveforms for the optimized design, with somewhat aggressive waveform shaping. Observe that, in the bottom picture, the

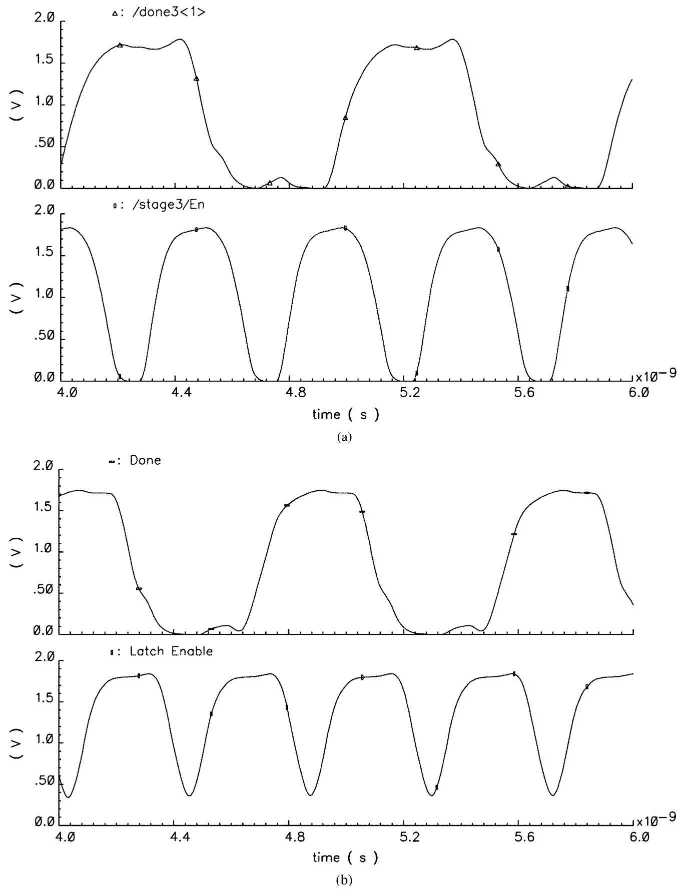

Fig. 13. Waveforms corresponding to the results of Table I. (a) With no (or moderate) waveform shipping. (b) With aggresive waveform shaping.

voltage swing is somewhat reduced, i.e., the latch enable signal is not completely deasserted before it is reasserted. This scenario corresponds to Fig. 8(d). The timing constraint of (8) (*data overrun*) still appears to be easily met: $t_{\text{Latch}} = 179$ ps, $t_{\text{logic}} = 0$ ps, and $t_{\text{XNOR}\downarrow} - t_{\text{XNOR}\uparrow} = 68$ ps. Strictly, however, the slew rates (i.e., rise and fall times) of the latch enable signal, which were not taken into account in (8), erode the available margin, thereby resulting in the reduced swing. The introduction of processing logic into the pipeline can help satisfy the timing constraint with more comfortable margins.

Power/Energy Consumption: Gate-level pipelining is designed for the highest throughput systems, which typically have greater power consumption.[3] However, a gate-level MOUSETRAP pipeline has several potential power advantages over a gate-level synchronous pipeline. First, only active stages in MOUSETRAP exhibit switching activity, whereas typically all stages in the synchronous pipeline have switching activity. While some clocked approaches employ *clock gating* [37] to reduce unwanted switching, there are overheads of additional design and verification complexity. Second, MOUSETRAP uses transparent D-latches, while most synchronous designs use more energy-expensive flip-flops or double latches. Finally, a basic MOUSETRAP has only a single gate of control logic per stage, and similar datapath loading as the synchronous pipeline, so it is expected that local control does not incur additional penalty over the synchronous implementation.

[3]Power consumption should be less for coarser-grain MOUSETRAP pipelines, in which most of the power will be dissipated in the standard synchronous-style bundled function blocks and their associated pipeline latches, and relatively little in the stage controllers.

TABLE II
POWER AND ENERGY CONSUMPTION OF TEN-STAGE FIFOs

Pipeline Design	Power (mW)	Energy/item/stage (pJ)
MOUSETRAP	30.8	1.49
PS0	26.3	5.20

Experimental results are now presented to compare the power and energy consumption of MOUSETRAP pipelines to another asynchronous style, and to show how MOUSETRAP can be significantly more energy efficient. In particular, the 16-bit unoptimized MOUSETRAP FIFO was evaluated both for power dissipation when it is running at maximum throughput, and also for energy consumed for one data item passing through one pipeline stage. Power dissipation is relevant when considering thermal issues and supply currents, whereas the latter figure is important to determine energy efficiency, which is critical for mobile and handheld systems that must operate on batteries with limited charge supplies.

Table II summarizes the simulation results for power and energy consumption. For comparison, a classic dual-rail dynamic pipeline design by Williams and Horowitz called PS0 [39] was also simulated. For each of the two styles, a 16-bit 10-stage FIFO was simulated. Since PS0 is dual-rail, its datapath actually consists of 32 wires. Data items were generated randomly for the pipelines such that each data bit has a probability of 0.5 that it changes between two consecutive data items. The middle column of the table shows the power consumed in milliwatts for each design. The last column shows the energy consumed in picojoules for each data item passing through one stage.

As seen in the table, MOUSETRAP has significantly better energy efficiency, though slightly higher power consumption. In particular, MOUSETRAP consumes 17% higher power than PS0 (30.8 mW instead of 26.3 mW). This higher power consumption in MOUSETRAP is solely because of its significantly higher throughput (2.1 GHz), i.e., it performs more "work" per second than the PS0 pipeline (0.51 GHz). In fact, MOUSETRAP consumes *71% lower energy* per item per stage, i.e., it is 3.5 times more energy efficient for the same computation. There are two reasons for the greater energy effiency of MOUSETRAP. First, it has lower switching activity because approximately only half the data bits switch between consecutive data items; in contrast, in dual-rail PS0, one rail of each bit must rise and precharge for each data item (i.e., 4× switching activity). Second, MOUSETRAP does not need completion detectors, which are required in PS0 and are a significant source of energy consumption. Taken together, these two factors would suggest an energy advantage for MOUSETRAP of greater than 4×; however, the benefit is slightly less (3.5×) because MOUSETRAP uses static latches, which are somewhat more energy expensive than the dynamic gates used in the PS0 pipeline.

VI. Conclusion

A new pipeline design style, MOUSETRAP, was introduced for high-throughput applications that use static logic datapaths. The pipeline uses simple structures for both latches and control, and an efficient and highly-concurrent event-driven protocol. The pipelines have only two simple one-sided timing constraints which in practice are usually easy to satisfy.

Optimizations are introduced at the logic and circuit levels, to further improve throughput. In particular, one style merges latches and logic gates through use of C^2MOS, resulting in a style that is particularly well-suited for gate-level pipelining. Another optimization removes critical inverter delays from the cycle time by implementing the control circuits in dual rail. Finally, a circuit-level optimization further speeds up critical events through a "waveform shaping" approach. As a result, in steady-state operation, the pipeline performance is comparable to that of wave pipelines, and yet the new pipelines are more robust and require much less design effort.

Acknowledgment

The authors would like to thank Prof. S. Furber and Prof. J. Garside of the University of Manchester, for pointing out potential issues of short-path timing violations when using the waveform shaping strategy.

References

[1] ARM, Cambridge, U.K., "ARM offers first clockless processor core," (2006). [Online]. Available: http://www.eetimes.com/news/latest/showArticle.jhtml?articleID=179101800

[2] M. Borah, R. M. Owens, and M. J. Irwin, "High-throughput and low-power DSP using clocked-CMOS circuitry," in *Proc. Int. Symp. Low-Power Design*, 1995, pp. 139–144.

[3] L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli, "The theory of latency insensitive design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 9, pp. 1059–1076, Sep. 2001.

[4] A. Chakraborty and M. R. Greenstreet, "Efficient self-timed interfaces for crossing clock domains," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, 2003, pp. 78–88.

[5] T. Chelcea and S. M. Nowick, "Robust interfaces for mixed-timing systems with application to latency-insensitive protocols," in *Proc. ACM/IEEE Design Autom. Conf.*, 2001, pp. 21–26.

[6] T.-A. Chu, "Synthesis of self-timed vlsi circuits from graph-theoretic specifications," Ph.D. dissertation, Lab. Comput. Sci., MIT, Cambridge, 1987.

[7] P. Day and J. V. Woods, "Investigation into micropipeline latch design styles," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 3, no. 2, pp. 264–272, Jun. 1995.

[8] A. Dooply and K. Yun, "Optimal clocking and enhanced testability for high-performance self-resetting domino pipelines," in *ARVLSI*, 1999, pp. 200–214.

[9] M. Ferretti and P. A. Beerel, "Single-track asynchronous pipeline templates using 1-of-N encoding," in *Proc. Design, Autom. Test Eur. (DATE)*, 2002, pp. 1008–1015.

[10] S. Furber, "Computing without clocks: Micropipelining the ARM processor," in *Proc. Asynchronous Digit. Circuit Design, Workshops Comput.*, 1995, pp. 211–262.

[11] S. B. Furber and P. Day, "Four-phase micropipeline latch control circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 4, no. 2, pp. 247–253, Jun. 1996.

[12] H. v. Gageldonk, D. Baumann, K. van Berkel, D. Gloor, A. Peeters, and G. Stegmann, "An asynchronous low-power 80C51 microcontroller," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst. (ASYNC)*, 1998, pp. 96–107.

[13] Handshake Solutions, Eindhoven, The Netherlands, "Home page" (2006). [Online]. Available: http://www.handshakesolutions.com

[14] D. Harris and M. Horowitz, "Skew-tolerant domino circuits," *IEEE J. Solid-States Circuits*, vol. 32, no. 11, pp. 1702–1711, Nov. 1997.

[15] O. Hauck, M. Garg, and S. A. Huss, "Two-phase asynchronous wave-pipelines and their application to a 2D-DCT," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits. Syst. (ASYNC)*, 1999, pp. 219–228.

[16] H. M. Jacobson, P. N. Kudva, P. Bose, P. W. Cook, S. E. Schuster, E. G. Mercer, and C. J. Myers, "Synchronous interlocked pipelines," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, 2002, pp. 3–12.

[17] J. Kessels, A. Peeters, and S.-J. Kim, "Bridging clock domains by synchronizing the mice in the mousetrap," in *Proc. Power Timing Modeling, Optimization Simulation (PATMOS), volume 2799 of Lecture Notes Comput. Sci.*, 2003, pp. 141–150.

[18] A. M. Lines, "Pipelined asynchronous circuits," Master's thesis, Dept. Comput. Sci., California Inst. Technol., Pasadena, 1998.

[19] W. Liu, C. T. Gray, D. Fan, W. J. Farlow, T. A. Hughes, and R. K. Cavin, "A 250-MHz wave pipelined adder in 2-μ m CMOS," *IEEE J. Solid-States Circuits*, vol. 29, no. 9, pp. 1117–1128, Sep. 1994.

[20] A. J. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth, and U. Cummings, "The design of an asynchronous MIPS R3000 microprocessor," in *Proc. ARVLSI*, 1997, pp. 164–181.

[21] C. Molnar, I. Jones, W. Coates, J. Lexau, S. Fairbanks, and I. Sutherland, "Two FIFO ring performance experiments," *Proc. IEEE*, vol. 87, no. 2, pp. 297–307, Feb. 1999.

[22] C. E. Molnar and I. W. Jones, "Simple circuits that work for complicated reasons," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst. (ASYNC)*, 2000, pp. 138–149.

[23] V. Narayanan, B. Chappell, and B. Fleischer, "Static timing analysis for self resetting circuits," in *Proc. ICCAD*, 1996, pp. 119–126.

[24] R. O. Ozdag and P. A. Beerel, "High-speed QDI asynchronous pipelines," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, 2002, pp. 13–22.

[25] S. Schuster, W. Reohr, P. Cook, D. Heidel, M. Immediato, and K. Jenkins, "Asynchronous interlocked pipelined CMOS circuits operating at 3.3–4.5 GHz," in *Proc. ISSCC*, 2000, pp. 292–293.

[26] C. L. Seitz, "System Timing," in *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980, ch. 7.

[27] M. Singh, "The design of high-throughput asynchronous pipelines," Ph.D. dissertation, Dept. Comput. Sci., Columbia Univ., New York, NY, 2001.

[28] M. Singh and S. M. Nowick, "Fine-grain pipelined asynchronous adders for high-speed DSP applications," in *Proc. IEEE Comput. Soc. Annu. Workshop VLSI*, 2000, pp. 111–118.

[29] M. Singh and S. M. Nowick, "High-throughput asynchronous pipelines for fine-grain dynamic datapaths," in *Proc. Intl. Symp. Adv. Res. Asynchronous Circuits Syst. (ASYNC)*, 2000, pp. 198–209.

[30] M. Singh and S. M. Nowick, "MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines," in *Proc. Int. Conf. Comput. Design (ICCD)*, 2001, pp. 9–17.

[31] M. Singh and M. Theobald, "Generalized latency-insensitive systems for single-clock and multi-clock architectures," in *Proc. Design, Autom. Test Eur. (DATE)*, 2004, pp. 1008–1013.

[32] M. Singh, J. A. Tierno, A. Rylyakov, S. Rylov, and S. M. Nowick, "An adaptively-pipelined mixed synchronous-asynchronous digital FIR filter chip operating at 1.3 gigahertz," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, 2002, pp. 84–95.

[33] I. Sutherland and S. Fairbanks, "GasP: A minimal FIFO control," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst. (ASYNC)*, 2001, pp. 46–53.

[34] I. Sutherland, B. Sproull, and D. Harris*, Logical Effort: Designing Fast CMOS Circuits*. San Mateo, CA: Morgan Kaufmann, 1999.

[35] I. E. Sutherland, "Micropipelines," *Commun. ACM*, vol. 32, no. 6, pp. 720–738, Jun. 1989.

[36] J. Tierno, A. Rylyakov, S. Rylov, M. Singh, P. Ampadu, S. Nowick, M. Immediato, and S. Gowda, "A 1.3 GSample/s 10-tap full-rate variable-latency self-timed FIR filter with clocked interfaces," in *Proc. Int. Solid State Circuits Conf.*, 2002, pp. 60–444.

[37] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, "Reducing power in high-performance microprocessors," in *Proc. ACM/IEEE Design Autom. Conf.*, 1998, pp. 732–737.

[38] C. van Berkel, M. Josephs, and S. Nowick, "Scanning the technology: Applications of asynchronous circuits," *Proc. IEEE*, vol. 87, no. 2, pp. 223–233, Feb. 1999.

[39] T. Williams, "Self-timed rings and their application to division," Ph.D. dissertation, Dept. Electr. Eng. Comput. Sci., Stanford Univ., Stanford, CA, 1991.

[40] D. Wong, G. De Micheli, and M. Flynn, "Designing high-performance digital circuits using wave-pipelining," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 12, no. 1, pp. 24–46, Jan. 1993.

[41] G. Yee and C. Sechen, "Clock-delayed domino for adder and combinational logic design," in *Proc. ICCD*, 1996, pp. 332–337.

[42] K. Yun, P. Beerel, and J. Arceo, "High-performance asynchronous pipeline circuits," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst. (ASYNC)*, 1996, pp. 17–28.

Montek Singh received the B.Tech. degree in electrical engineering from IIT Delhi, Delhi, India, and the Ph.D. degree in computer science from Columbia University, New York, NY, in 2002.

He is an Assistant Professor with the Department of Computer Science, the University of North Carolina, Chapel Hill. His research interests include the area of asynchronous circuits and systems, especially circuits and synthesis tools for the design of high-speed pipelined systems. In 2005, he was brought onto the DARPA CLASS Program (led by Boeing), to develop, in collaboration with Handshake Solutions, an industrial-strength automated synthesis flow for designing high-speed pipelined asynchronous systems. His work has been transferred to industries, including IBM, Boeing, and Handshake Solutions (a Philips subsidiary).

Dr. Singh was a recipient of a Best Paper Award at the 2000 IEEE Async Symposium, a Best Paper Finalist Nomination at the 2002 Async Symposium, and an IBM Faculty Award in 2004. He was a Program Committee Co-Chair for the Async'07 Symposium and the FMGALS'05 Workshop.

Steven M. Nowick received the B.A. degree from Yale University, New Haven, CT, in 1976, and the Ph.D. degree in computer science from Stanford University, Stanford, CA, in 1993.

He is an Associate Professor with the Department of Computer Science and Electrical Engineering, Columbia University, New York, NY. His main research interest is on CAD tools, as well as design methods, for the synthesis, analysis, and optimization of asynchronous and mixed-timing digital systems. In 2005, was brought onto DARPA's CLASS project, headed by Boeing, with participation of a Philips-based startup (Handshake Solutions), to create a commercially-viable CAD tool flow for designing asynchronous systems. He was also a co-founder of the IEEE Async Symposia series.

Dr. Nowick was a recipient of a National Science Foundation (NSF) Faculty Early Career (CAREER) Award (1995), an Alfred P. Sloan Research Fellowship (1995), an NSF Research Initiation Award (RIA) (1993), two Best Paper Awards, one at the 1991 International Conference on Computer Design and the other at the 2000 IEEE Async Symposium, and two medium-scale NSF ITR Awards for asynchronous research in 2000. He was Program Committee Co-Chair of Async'94 and Async'99 and General Co-Chair of Async'05, and was Program Chair of the IWLS'02 Workshop. He is currently an Associate Editor of the IEEE Transactions on Computer-Aided Design and of IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS.